A Transformational Method for Verifying Safety Properties in Real-Time Systems*

Matthew K. Franklin and Armen Gabrielian

Technical Report 89-12

July 1989

THOMSON-CSF, INC.
PACIFIC RIM OPERATIONS

630 Hansen Way, Suite 250
Palo Alto, California 94304

4

A Transformational Method for Verifying Safety Properties in Real-Time Systems*

Matthew K. Franklin and Armen Gabrielian

Technical Report 89-12

July 1989

* To be presented at the 10th Real-Time Systems Symposium, Los Angeles, CA, Dec. 7-9, 1989. The work reported in this paper was supported in part by the Office of Naval Research under Contract No. N00014-89-C-0022.

To be presented at the 10th Real-Time Systems Symposium, Los Angeles, CA, Dec. 7-9, 1989

A Transformational Method for Verifying Safety Properties in Real-Time Systems*

Matthew K. Franklin and Armen Gabrielian

Thomson-CSF, Inc.
630 Hansen Way, Suite 250
Palo Alto, CA 94304

Abstract

The behavior of systems with hard real-time constraints can be specified in terms of Hierarchical Multi-State (HMS) abstract machines, which are generalizations of finite-state automata. In this paper, a two-step method is presented for verifying that *safety properties* are not violated by an HMS specification of a system. In the first step, the safety verification question is recast as a reachability problem in an extension of the HMS machine. In the second step, reachability is determined by the use of correctness-preserving and partial correctness-preserving transformations. The method is shown to be complete, and it is illustrated by verifying that a safety property holds for a simple railroad-crossing system if all of its deadlines are met.

1. Introduction

Hierarchical Multi-State (HMS) machines are a type of automata that can be used for the specification of complex real-time systems [Ga-88], as well as for the causal modeling of physical systems [Ga-87]. Key features of HMS machines that provide a means for dealing with complexity are (i) multiple active states, (ii) simultaneous firing of multiple transitions, (iii) hierarchies, (iv) object-carrying tokens, and (v) non-deterministic transitions that allow the specification of entire classes of related systems. Another feature of HMS machines is a special language for defining temporal constraints on transitions.

The goal of this paper is to present a method for verifying that a system, as specified by an HMS machine, does not violate a set of "safety properties," which are requirements that define undesirable situations for the system. The method first represents the requirements as new states in the HMS machine, thus converting a problem of logical satisfaction into a problem of state unreachability. Next, transformations are applied to the HMS machine that alter its structure, while maintaining critical aspects of its behavior. Verification is complete when the machine has been transformed into one for which unreachability is obvious (e.g., no transitions leading into the given state). A preliminary and informal presentation of this method appeared in [Ga-88], where it was used to prove that a protocol for a two-processor mutual exclusion protocol is collision-free.

The method of requirement representation outlined above is analogous to the representation of propositional logic "facts" in Petri Nets (e.g., [Re-85]). In an HMS machine, however, temporal logic "safety properties" [La-77], including properties which mandate that hard deadlines are met, can be represented. The use of correctness-preserving transformations

* The work reported in this paper was supported in part by the Office of Naval Research under Contract No. N00014-89-C-0022.

is common in formal software development [Pa-83], and rewrite rules are a standard part of the syntactic theories of deduction and computation [Hu-85]. Transformations on classical finite-state automata are also common, going back at least as far as the proof of the equivalence of non-deterministic and deterministic automata [Ra-59]. The modular construction of Petri nets through stepwise refinement has been widely explored (beginning with [Va-79], and more recently in [Vo-89]). Non-transformational approaches to proving that formal specifications meet safety requirements include the construction of "refinement mappings" [Ab-88], the discharging of "variance and invariance proof obligations" [Al-85], and the construction of "computation graphs" [Ja-88].

In Sections 2 of this paper, the structure and execution of HMS machines are defined formally. In Section 3, the concept of transformations on HMS machines is introduced, along with a specific set of transformations that are shown to be complete for verification purposes. In Section 4, the method of representing safety properties as extended states of an HMS machine is presented. A demonstration of the proof method for a simple railroad-crossing system with hard deadlines is presented in Section 5 (this example was originally used in [Ja-88]). The summary and conclusions appear in Section 6.

2. HMS Machines

As noted in the Introduction, multiple states may be "active" at any moment and many transitions may "fire" simultaneously in an HMS machine. The states of an HMS machine may be organized hierarchically, although hierarchies will not be considered in this paper (see [Ga-88]). A transition can fire only if certain logical and temporal conditions are satisfied. If its states correspond to the relevant facts about a system, and its transitions correspond to the system dynamics, then the HMS machine constitutes a "specification" of that system. Given such a specification, and assuming a model of time as a discrete, linear and unbounded ordering, the "executions" of the HMS machine correspond to the possible behaviors of the system.

There is actually a succession of increasingly powerful classes of HMS machines that can be defined by varying the types of states in the machine (e.g., by introducing "tokens" into them). In this paper, we will consider only the simplest of these classes, in which a state is either marked or unmarked. Thus, the class of machines considered in this paper is a variation of a subclass of the "HMS-0" machines of [Ga-88], where informal operational definitions were given. The more formal definitions of this paper are key ingredients for proving the consistency and completeness of our transformational approach to verification.

2.1. Structure of HMS Machines

In this section, the definition of the *structure* of HMS machines is presented. The purpose of the various components of an HMS machine will become clear when the *execution* of HMS machines is described in Section 2.2.

Informally, an HMS machine is defined as a set of states (denoted by S), together with deterministic and non-deterministic transitions among these states (denoted Γ_D and Γ_N, respectively). The following three definitions introduce basic concepts.

Definition 1 (Time Expression): τ is a "time expression" if τ is either $<t_1, t_2>$, $<t_1, t_2>!$, or $[t_1, t_2]$, where t_1 and t_2 are integers, with $t_1 \leq t_2 \leq 0$. When $t_1 = t_2 = t$, these three time expressions may be written as t, t!, and t respectively.

Definition 2 (Control): c is a "control" over S if τ is a time expression and (a) c is (s, τ) or $(\neg s, \tau)$, for some state s; or (b) c is (id, τ) or $(\neg id, \tau)$, for some transition label id (see Definition 3). Controls of type (a) are called "state-based controls," and controls of type (b) are called "transition-based" controls.

In practice, only state-based controls are necessary for system specification. The transition-based controls, which were not a part of HMS definitions in [Ga-88], greatly simplify the ideas of transformational proofs in Section 3.

Definition 3 (Transition): γ is a "transition" over S if γ is of the form

id: (PRIMARIES) (CONTROLS) --> (CONSEQUENTS)

where id is a "label," PRIMARIES is a (possibly empty) subset of S, CONTROLS is a (possibly empty) set of controls over S, and CONSEQUENTS is a (possibly empty) subset of S. These three sets will be denoted PRIMS(γ), CTRLS(γ) and CNSQS(γ), respectively.

Definition 4 (HMS Machine): An "HMS machine" is a triple $H = (S, \Gamma_D, \Gamma_N)$, where S is a set of states, and Γ_D and Γ_N are (possibly empty) sets of transitions over S. H is "state-controlled" if its transitions use only state-based controls.

State-controlled HMS machines are equivalent to HMS-0 machines of [Ga-88] without hierarchies, special states ("initial," "final," and "external"), or "future delays."

These definitions can be illustrated by the state-controlled HMS machine of Figure 1, which defines the operation of a user-controlled mixer, where

S = {Switch-ON, Switch-OFF, MIXING, IDLE}
Γ_D = {w: (MIXING) ((¬Switch-ON, -1) (MIXING, [-10, 0])) --> (IDLE),
x: (IDLE) ((Switch-ON, -1) (MIXING, <-30, 0>)) --> (MIXING)}
Γ_N = {y: (Switch-ON) ( ) --> (Switch-OFF),
z: (Switch-OFF) ( ) --> (Switch-ON)}.

In the graphic notation of Figure 1, (1) states are represented as boxes (2) transitions are represented as dark arrows from primaries to consequents, and (3) controls are denoted by thin arrows. Non-determinism is indicated by an asterisk at the head of a transition arrow, time expressions appear next to the symbol Ⓣ, and negation is denoted by the standard symbol used in VLSI design. Transition labels such as w, x, y and z are normally not shown in the graphic representation of HMS machines.

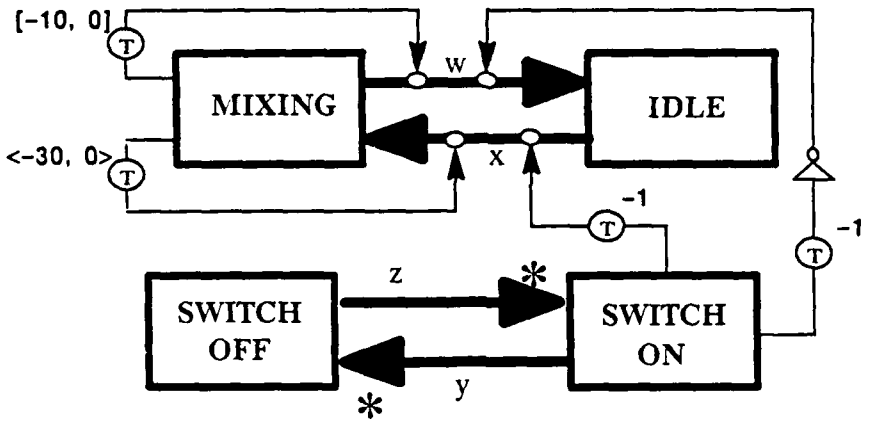

Figure 1. Mixer HMS Machine

2.2. Execution of HMS Machines

The legal execution of HMS machines is described in this section. Informally, a machine executes by "firing" some of its transitions at each moment, which alters the values of some or all of the primary states and consequent states at the next moment. Starting from a description of initial conditions, an execution of an HMS machine is determined by a sequence of sets of transitions, indicating what happens at each successive moment. Due to the non-deterministic transitions, there may be a large set of possible executions for a given machine under given initial conditions. As noted in [Ga-88], non-determinism is a powerful tool for defining a "generic" specification for an entire *class* of systems that can be refined using methods outlined therein.

We begin with the notion of "marking," which defines the status of an HMS machine at the present and at all moments in the infinite past. The assumption of an infinite past will simplify later definitions, although only finite histories are needed in practice.

Definition 5 (Marking): M is a "marking" of an HMS machine $H = (S, \Gamma_D, \Gamma_N)$ if M is a mapping from $(S \cup \Gamma_D \cup \Gamma_N) \times \{0, -1, -2, ...\}$ to {T, F}. If M(s, i) = T (F), then the state s is said to be "marked" ("unmarked") or "true" ("false") at time i. If M(γ, i) = T (F), then the transition γ is said to have "fired" ("not fired") at time i.

An HMS machine is "executable," in the sense that a sequence of successive markings may be generated from an initial marking. At any moment of time, some transitions of an HMS machine will be "enabled." The firing of all enabled deterministic transitions and a subset (possibly empty) of enabled non-deterministic transitions at that moment yields a new marking at the *next* moment. This process is formalized by Definitions 6-10:

Definition 6 (Control Satisfaction): The control c is "satisfied" for marking M if

(i) c is $(x, <t_1, t_2>)$ and $M(x, t_3) = T$ for some t_3 s.t. $t_1 \leq t_3 \leq t_2$.

(ii) c is $(x, [t_1, t_2])$ and $M(x, t_3) = T$ for every t_3 s.t. $t_1 \leq t_3 \leq t_2$.

(iii) c is $(x, <t_1, t_2>!)$, $M(x, t_3) = T$ for some t_3 s.t. $t_1 \leq t_3 \leq t_2$, and $M(x, t_1 - 1) = F$.

(iv) c is $(\neg x, \langle t_1, t_2 \rangle)$ and $M(x, t_3) = F$ for some t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(v) c is $(\neg x, [t_1, t_2])$ and $M(x, t_3) = F$ for every t_3 s.t. $t_1 \leq t_3 \leq t_2$.
(vi) c is $(\neg x, \langle t_1, t_2 \rangle!)$, $M(x, t_3) = F$ for some t_3 s.t. $t_1 \leq t_3 \leq t_2$, and $M(x, t_1 - 1) = T$.

The different time expressions have informal names that are consistent with the definition of control satisfaction: $\langle t_1, t_2 \rangle$ is called a "sometime-delay," $[t_1, t_2]$ is called an "always-delay," and $\langle t_1, t_2 \rangle!$ is called a "sometime-change-delay."

Definition 7 (Transition Enablement): The transition γ is "enabled" for marking M if (1) $M(s, 0) = T$ for all s in PRIMS(γ), and (2) c is satisfied for M for all c in CTRLS(γ).

For example, in the HMS machine MIXER from Section 2.1, the transition from IDLE to MIXING is enabled if (a) IDLE is currently true, (b) SWITCH-ON was true at the previous moment, and (c) MIXING was true within the past thirty moments (this constraint would be consistent with a model of a cement mixer, for which mixing would become impossible after a long period of idling).

For convenience, we define the following sets for $H = (S, \Gamma_D, \Gamma_N)$, and marking M:

$$\text{D-ENAB}(H, M) = \{\gamma \mid \gamma \in \Gamma_D \text{ and } \gamma \text{ enabled for } M\}$$
$$\text{N-ENAB}(H, M) = \{\gamma \mid \gamma \in \Gamma_N \text{ and } \gamma \text{ enabled for } M\}.$$

Definition 8 (Firing Set): The set of transitions Γ is a "firing set" of $H = (S, \Gamma_D, \Gamma_N)$ for marking M if $\Gamma = \text{D-ENAB}(H, M) \cup \Gamma'$, where $\Gamma' \subseteq \text{N-ENAB}(H, M)$.

Definition 9 (Next Marking): If M is a marking of an HMS machine H, and if Γ is a firing set of H for M, then the marking "after M via Γ" is denoted by $M[\Gamma]$, and is given by

$M[\Gamma](s, 0) = T$ for every state s in CNSQS(Γ)
$M[\Gamma](s, 0) = F$ for every state s in PRIMS(Γ) but not CNSQS(Γ)
$M[\Gamma](s, 0) = M(s, 0)$ for every state s not in either PRIMS(Γ) or CNSQS(Γ)
$M[\Gamma](\gamma, 0) = T$ for every transition $\gamma \in \Gamma$
$M[\Gamma](\gamma, 0) = F$ for every transition $\gamma \notin \Gamma$
$M[\Gamma](x, t) = M(x, t + 1)$ for every $x \in S \cup \Gamma_D \cup \Gamma_N$, and every time $t < 0$.

Definition 10 (HMS Execution): If H is an HMS machine, and if M_0 is a marking of H, then an "execution of H from M_0" is a sequence $[M_0, M_1, M_2, \ldots]$ of markings such that $M_{i+1} = M_i[\Gamma^i]$ for some firing set Γ^i of H for M_i, for each $i \geq 0$. The set of all executions of H from M_0 is denoted by $\mathfrak{E}(H, M_0)$.

For example, one possible marking M of the HMS machine MIXER from Section 2.1 is

M(Switch-OFF, i) = M(IDLE, i) = T for all $i \leq 0$.
M(Switch-ON, i) = M(MIXING, i) = F for all $i \leq 0$.
M(w, i) = M(x, i) = M(y, i) = M(z, i) = F for all $i \leq 0$.

The marking after M via firing set { z: (Switch-OFF) ( ) --> (Switch-ON) } is given by

$M[\Gamma](\text{Switch-ON}, 0) = T$; $M[\Gamma](\text{Switch-ON}, i) = F$ for all $i < 0$
$M[\Gamma](\text{Switch-OFF}, 0) = F$; $M[\Gamma](\text{Switch-OFF}, i) = T$ for all $i < 0$
$M[\Gamma](\text{IDLE}, i) = T$ for all $i \leq 0$; $M(\text{MIXING}, i) = F$ for all $i \leq 0$
$M(z, 0) = T$; $M(z, i) = F$ for all $i < 0$; $M(w, i) = M(x, i) = M(y, i) = F$ for all $i \leq 0$.

3. Transformations on HMS Machines

In this section, we define local transformations on HMS machines that modify the structure of a machine while maintaining significant aspects of its behavior. The repeated application of such transformations can lead to a machine with a very simple structure, for which the determination of a desired condition is trivial.

In Section 3.1, both "correctness-preserving" and "partial-correctness-preserving" transformations are defined. In Section 3.2, the consistency and completeness of these transformations is demonstrated. The completeness proof is constructive, although for practical examples a more judicious choice of transformations is needed to make the process manageable. The application of our method will be illustrated in Section 5 by verifying a safety condition for a railroad-crossing system.

3.1. Definition of Transformations

Before formalizing the notion of correctness-preserving transformation, we introduce four preliminary definitions. Definition 11 describes the shifting of time expressions forward or backward in time. Every control, except a sometime-change control, has a complement, which is given in Definition 12. The 1-invariants of Definition 13, which describe an unchanging property of the markings of a subset of states, are analogous to S-invariants from Petri Net Theory [Re-85]. Finally, Definition 14 introduces the notion of marking consistency, which indicates the plausibility of the past history of an HMS machine.

Definition 11 (Time Expression Shift): The "shift of time expression τ by d" is denoted shift(τ, d), and is given by (a) shift($\langle t_1, t_2\rangle$, d) = $\langle t_1 + d, t_2 + d\rangle$, (b) shift($[t_1, t_2]$, d) = $[t_1 + d, t_2 + d]$, and (c) shift($\langle t_1, t_2\rangle!$, d) = $\langle t_1 + d, t_2 + d\rangle!$.

Definition 12 (Control Complement): If c is a sometime-control or always-control, then the "complement of c" is denoted comp(c), and is given by (a) comp((x, $\langle t_1, t_2\rangle$)) = ($\neg$x, $[t_1, t_2]$), (b) comp(($\neg$x, $\langle t_1, t_2\rangle$)) = (x, $[t_1, t_2]$), (c) comp((x, $[t_1, t_2]$)) = ($\neg$x, $\langle t_1, t_2\rangle$), and (d) comp(($\neg$x, $[t_1, t_2]$)) = (x, $\langle t_1, t_2\rangle$). It follows from the definition of control satisfaction that every marking satisfies exactly one of c and comp(c).

Definition 13 (1-Invariance): Let $H = (S, \Gamma_D, \Gamma_N)$. Then $S' \subseteq S$ is a "1-invariant subset of H" if, for every initial marking M_0 and every M in every execution in $\mathfrak{E}(H, M_0)$,

$$||\{s' \mid s' \in S' \wedge M_0(s', 0) = T\}|| = 1 \Rightarrow ||\{s' \mid s' \in S' \wedge M(s', 0) = T\}|| = 1.$$

Although this paper will not consider the general problem of 1-invariant discovery, we note one simple type: when the transitions into and out of states in S' form a cycle. Two 1-invariants of this type will be needed in the proof of the example in Section 5.2.

Definition 14 (Consistent Marking): A marking M of an HMS machine H is "consistent to k in the past" if there is a marking M* of H such that M is the (k+1)st marking in some execution of H from M*. M is "consistent" if it is consistent to k in the past for all $k \geq 0$.

We now introduce four basic classes of correctness-preserving transformations: Delay Change modifies the time expression of a control of a transition; Case Split divides a transition with respect to mutually exclusive facts; Control Addition augments a transition with some logical consequence of its existing controls; Transition Deletion removes a transition with contradictory controls. One member of each transformation class is presented in Figure 2; formal definitions of four or five members of each class follow.

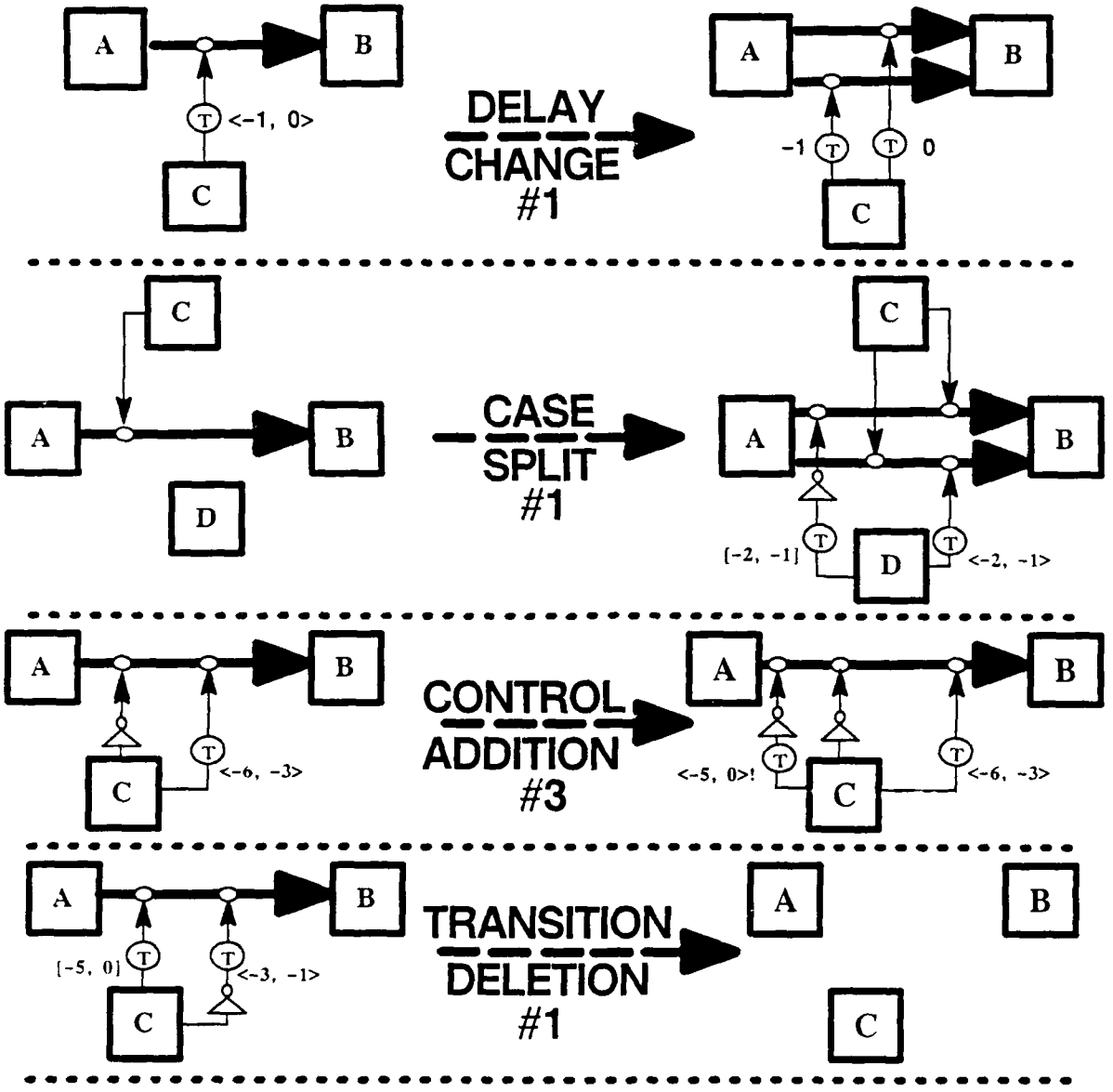

Figure 2. Examples of Correctness-Preserving Transformations

Delay Change:

(1) If transition γ has control $c = (x, <t_1, t_2>)$, where $t_1 < t_2$, split γ into two transitions γ_1 and γ_2 such that (a) γ_1 differs from γ only in that c is replaced by $c_1 = (x, <t_1, t_3>)$, (b) γ_2 differs from γ only in that c is replaced by $c_2 = (x, <t_3+1, t_2>)$, and (c) $t_1 \leq t_3 \leq t_2$.

(2) If transition γ has control $c = (x, [t_1, t_2])$, where $t_1 < t_2$, split c into two controls $c_1 = (x, [t_1, t_3])$ and $c_2 = (x, [t_3+1, t_2])$, where $t_1 \leq t_3 \leq t_2$.

(3) If transition γ has control $(\neg s, <t_1, t_2>)$ or $(\neg s, [t_1, t_2])$, and no transition has primary s, then add the control $(\neg s, [-\infty, t_1])$.

(4) If transition γ has control $(s, <t_1, t_2>)$ or $(s, [t_1, t_2])$, and no transition has consequent s, then add the control $(s, [-\infty, t_1])$.

Case Split:

(1) Split transition γ into two transitions γ_1 and γ_2 such that control c is added to γ_1, and control comp(c) is added to γ_2

(2) If transition γ has control $c = (s, <t_1, t_2>!)$, and if $\{\gamma'_1, \ldots, \gamma'_n\}$ is the set of all transitions with consequent s, split γ into $\gamma_1, \ldots, \gamma_n$, where γ_i is identical to γ except that it has the two additional controls $(\gamma'_i, <t_1, t_2>)$ and $(\neg s, t_1 - 1)$.

(3) If transition γ has control $c = (\neg s, <t_1, t_2>!)$, and if $\{\gamma'_1, \ldots, \gamma'_n\}$ is the set of all transitions with primary s, split γ into $\gamma_1, \ldots \gamma_n$, where γ_i is identical to γ, except that it has the two additional controls $(\gamma'_i, <t_1, t_2>)$ and $(s, t_1 - 1)$.

(4) If transition γ has control $(\gamma', <t_1, t_2>)$, s is a primary of γ', and $\{\gamma'_1, \ldots, \gamma'_n\}$ is the set of all transitions with consequent s, then split γ into $\gamma_0, \gamma_1, \ldots \gamma_n$, where $(\neg s, <t_1, t_2>)$ is added to γ_0, and $(\gamma'_i, <t_1, t_2>)$ is added to γ_i for $1 \leq i \leq n$.

Control Addition:

(1) If transition γ has control (γ', τ), then add to γ

(a) $(x, \text{shift}(\tau', t-1))$, if (x, τ') is a control of γ' and $\tau = t$.

(b) $(p, \text{shift}(\tau, -1))$, if p is a primary of γ'.

(c) $(x, <t'_1 + t_1 - 1, t'_2 + t_2 - 1>)$, if $(x, <t'_1, t'_2>) \in \text{CTRLS}(\gamma')$, $\tau = <t_1, t_2>$.

(d) $(x, <t'_1 + t_1 - 1, t'_2 + t_2 - 1>!)$, if $(x, <t'_1, t'_2>!) \in \text{CTRLS}(\gamma')$, $\tau = <t_1, t_2>$

(2) If transition γ has control $(\gamma', <t_1, t_2>)$, and if s is a consequent of γ', then add control $(s, <t_1, t_2>)$ to γ.

(3) If transition γ has controls (s, τ) and $(\neg s, \tau')$, then 46 transformations similar to the following two cases can be defined:

(a) if $\tau = [t_1, t_2]$, $\tau' = [t_1', t_2']$, and $t_2 < t_1'$, then add control $(\neg s, <t_2+1, t_1'>!)$

(b) if $\tau = [t_1, t_2]$, $\tau' = <t_1', t_2'>$, and $t_2 < t_1'$, then add control $(\neg s, <t_2+1, t_2'>!)$.

(4) Add any control that reflects a 1-invariant (Definition 13), e.g., if transition γ has control (s, τ), and $\{s, s'\}$ is a 1-invariant, then control $(\neg s', \tau)$ can be added.

(5) If (p, d) ∈ CTRLS(γ) for every p ∈ PRIMS(γ'), and (x, shift(τ, d)) ∈ CTRLS(γ) for every (x, τ) ∈ CTRLS(γ') for some deterministic γ' and d ≤ 0, then add (γ', d+1) to γ.

Transition Deletion: Remove transition γ, if
(1) γ has conflicting controls (e.g., (s, [t_1, t_2]) and (¬s, <t_1', t_2'>), $t_1 \leq t_1' \leq t_2' \leq t_2$).
(2) γ has control (s, <t_1, t_2>!) but no transition has s as a consequent.
(3) γ has control (¬s, <t_1, t_2>!), but no transition has s as a primary.
(4) γ has controls that conflict with some 1-invariant (Definition 13).
(5) Some γ', of same type, with same PRIMS and CNSQS, is enabled whenever γ is.

Note that Transition Deletion #1 and Control Addition #3 are special cases of a class of rewrites for transitions with controls with the same state (e.g., {(s, [t_1, t_2]), (s, [t_3, t_4])} ⇒ {(s, [t_1, t_4])} when $t_1 \leq t_3 \leq t_2 \leq t_4$)). There are 108 cases in this class, all of which are straightforward consequences of the definition of control satisfaction (Definition 6).

This list of correctness-preserving transformations is by no means exhaustive. However, the eighteen transformations will be proven to be both consistent and complete in Section 3.2, making them an adequate tool for answering the type of requirement satisfaction problems that will be introduced in Section 4.

There are transformations which are useful in constructing proofs, even though they do not preserve all possible executions of an HMS machine under all possible initial markings. One such "partial-correctness-preserving transformation," called "Delay Sharpening," is illustrated in Figure 3, and is formally defined as follows:

Delay Sharpening: If transition γ has primaries p_1, ..., p_n and controls c_1, ..., c_m, then obtain γ_1, ..., γ_n, γ_1', ..., γ'_m from γ by adding the following controls to copies of γ:
(a) the control (p_i, 0!) is added to γ_i
(b) the control (x, t!) is added to γ'_i if c_i is (x, t).
(c) the control (x, t_2!) is added to γ'_i if c_i is (x, <t_1, t_2>)
(d) the controls (x, t_1!) and (x, [t_1+1, t_2]) are added to γ'_i if c_i is (x, [t_1, t_2])

Figure 3. Partial-Correctness-Preserving Transformation

3.2. Consistency and Completeness of Transformations

The proof that the eighteen correctness-preserving transformations cannot affect the possible executions of an HMS machine relies on the following Lemma. To state the lemma, the

definition of the restriction of a marking or an execution is needed:

Definition 15 (Restriction): Let $H = (S, \Gamma_D, \Gamma_N)$, let $\Psi \subseteq S \cup \Gamma_D \cup \Gamma_N$, and let $E = [M_0, M_1, ...]$ be an execution in $\mathcal{E}(H, M_0)$. Then the following "restrictions" can be defined:

$M_0 \uparrow \Psi$ is the marking function M_0 restricted to $\Psi \times \{0, -1, -2, ...\}$

$E \uparrow \Psi$ is the sequence $[M_0 \uparrow \Psi, M_1 \uparrow \Psi, ...]$

$\mathcal{E}(H, M_0) \uparrow \Psi$ is the set $\{E \uparrow \Psi \mid E \in \mathcal{E}(H, M_0)\}$

Lemma 1 (Equivalent Execution): Let $H = (S, \Gamma_D, \Gamma_N)$ and $H' = (S, \Gamma'_D, \Gamma'_N)$, and let M and M' be markings of H and H' such that $M \uparrow S = M' \uparrow S$. Suppose that, for any enabled transition γ from either machine and of either type (deterministic or non-deterministic), there is an enabled transition γ' of the same type in the other machine, such that $PRIMS(\gamma) = PRIMS(\gamma')$ and $CNSQS(\gamma) = CNSQS(\gamma')$. Then $\mathcal{E}(H, M) \uparrow S = \mathcal{E}(H', M') \uparrow S$.

Proof (sketch): Suppose that $\Gamma = D\text{-}ENAB(H, M) \cup \Gamma_1$ is a firing set of H w.r.t. M, where $\Gamma_1 \subseteq N\text{-}ENAB(H, M)$. By the conditions of the Lemma, there is a firing set $\Gamma' = D\text{-}ENAB(H', M') \cup \Gamma'_1$ such that $PRIMS(\Gamma) = PRIMS(\Gamma')$ and $CNSQS(\Gamma) = CNSQS(\Gamma')$. From Definition 9, then, we have that $M[\Gamma] \uparrow S = M'[\Gamma'] \uparrow S$, and thus $\mathcal{E}(H, M) \uparrow S \subseteq \mathcal{E}(H', M') \uparrow S$. The proof of the other direction is identical. □

Using the Equivalent Execution Lemma, the following theorem demonstrates the consistency of the correctness-preserving transformations, in the sense that the range of behavior of any transformed HMS machine is identical to that of the original HMS machine.

Theorem 1 (Transformation Consistency): Let $H = (S, \Gamma_D, \Gamma_N)$, and let H' be derived from H by one application of a Delay Change, Case Split, Control Addition or Transition Deletion transformation from Section 3.1. Let M and M' be markings of H and H' such that $M \uparrow S = M' \uparrow S$. Then $\mathcal{E}(H, M) \uparrow S = \mathcal{E}(H', M') \uparrow S$.

Proof (sketch): The proof will be given for one case; the other seventeen cases are similar.

Let γ be a transition in H with control $c = (s, <t_1, t_2>)$, $t_1 < t_2$, and let H' be derived from H by a Delay Change #1 applied to γ. Then H' is identical to H, except that γ is replaced by γ_1 and γ_2, where γ_1 has c replaced by $(s, <t_1, t_3>)$, and γ_2 has c replaced by $(s, <t_3+1, t_2>)$, $t_1 \leq t_3 \leq t_2$. Let M and M' be markings of H and H' such that $M \uparrow S = M' \uparrow S$.

Then, M satisfies control $c = (s, <t_1, t_2>) \Leftrightarrow$

$M(s, t') = T$ for some t' s.t. $t_1 \leq t' \leq t_2 \Leftrightarrow$

$M(s, t') = T$ for some t' s.t. $t_1 \leq t' \leq t_3$ or $M(s, t') = T$ for some t' s.t. $t_3 < t' \leq t_2 \Leftrightarrow$

$M'(s, t') = T$ for some t' s.t. $t_1 \leq t' \leq t_3$ or $M'(s, t') = T$ for some t' s.t. $t_3 < t' \leq t_2 \Leftrightarrow$

M' satisfies control $c_1 = (s, <t_1, t_3>)$ or M' satisfies control $c_2 = (s, <t_3+1, t_2>)$

But then (M satisfies γ) $\Leftrightarrow$ (M' satisfies γ_1 or M' satisfies γ_2). Since γ, γ_1 and γ_2 have the same primaries and consequents, and are all of the same type, Lemma 1 implies that $\mathcal{E}(H, M) \uparrow S = \mathcal{E}(H', M') \uparrow S$. □

The next theorem demonstrates that nine of the eighteen transformations are sufficient to demonstrate unreachability in *any* HMS machine of the type defined in Section 2:

Theorem 2 (Transformation Completeness): If transition γ is not enabled in any consistent marking of HMS machine H, then γ can be deleted by performing a finite sequence of Case Split #1–4, Control Addition #1, 2, 3 and 5, and Transition Deletion #1 transformations.

Proof (sketch): Let -d be the smallest number appearing in any time expression of any transition in H. By a succession of Case Splits #1, all possible executions from $2^{|S|(d+1)}$ moments ago to the present moment can be represented as new controls. Then, by a combination of Control Additions #2 and #3, Case Splits #1, #2, #3 and #4, and Transition Deletions #1, all possible markings consistent with the executions from $2^{|S|(d+1)}$ moments ago to the present moment can be represented. Also, by a combination of Control Additions #1 and #5, all necessary preconditions of those executions can be represented. For each resulting transition γ', there are two possibilities: a contradiction exists between two controls, or no contradiction exists. In the first case, a Transition Deletion #1 removes γ'. In the second case, a consistent marking M^* can be constructed which enables γ. The key to this construction is the Pigeonhole Principle, which guarantees that, in a marking of length $2^{|S|(d+1)}+1$, there must be two moments i and j, with $-2^{|S|(d+1)} \leq i < j \leq 0$, such that the marking of all states at i+k agrees with the marking of all states at j+k, for every $-d \leq k \leq 0$. Then, if M^* agrees with the possible marking represented on γ' from i to 0, and repeats the marking from i to j back into the infinite past, it can be shown that M^* is consistent and that it enables γ, which is a contradiction. □

Although this proof is constructive, the number of steps it would require is prohibitive. Stronger proofs probably exist that would give a smaller bound on the number of steps. In practice, however, with proper selection and ordering of transformations, the complexity of a verification is manageable. For example, the proof of the safety of the railroad-crossing example in Section 5.2 requires 104 transformation steps, many of which fall into simple and natural sequences of operations.

Note that the completeness result does not apply to the problem of unreachability with respect to properties of an initial marking. However, this case can be covered whenever the initial conditions are representable as additional controls on γ.

Although the Delay Sharpening transformation does not maintain complete behavioral equivalence, it can preserve an important property of HMS machines -- "unreachability:"

Definition 16 (Unreachability): If $H = (S, \Gamma_D, \Gamma_N)$ is an HMS machine, and if M_0 is a marking of H, then a state s in S is "unreachable from M_0" if M(s, 0) = F for every marking M in every execution in $\mathcal{E}(H, M_0)$.

The second transformation theorem gives the conditions under which a Delay Sharpening transformation may be used to preserve unreachability in an HMS machine:

Theorem 3 (Partial-Correctness Transformation Consistency): Let $H' = (S, \Gamma'_D, \Gamma'_N)$ be derived from $H = (S, \Gamma_D, \Gamma_N)$ by a single Delay Sharpening transformation to a transition γ. Let M and M' be markings of H and H' respectively, such that $M\uparrow S = M'\uparrow S$. If the state s is unreachable in H' from M', for some $s \in CNSQS(\gamma)$, then s is unreachable in H from M.

Proof (sketch): If s is reachable in H from M without firing γ, then the same execution is possible for H' from M'. Otherwise, if s is reachable in H from M, then γ must be fired, and thus γ must be enabled, and thus γ must be enabled for a first time. The execution of H from M up to the first enablement of γ can be duplicated in H' from M', and at that moment at least one of the new transitions must be enabled. □

4. Representation of Requirements in HMS Machines

In this section, a method will be presented for representing system requirements as new states in an HMS machine. In particular, this method can represent "safety properties," which say, informally, that "something bad never happens." These requirements, which include hard deadlines, can be associated with new states of an HMS machine, so that the new states are unreachable if and only if the requirements are guaranteed to hold. It is advantageous to have system requirements given in the same HMS machine as the system specification, because it reduces a logical condition (satisfaction) to an execution condition (reachability). As was shown in Section 3, correctness-preserving transformations can answer questions of unreachability, and hence can be used to verify that a system specification meets desired safety properties. Lastly, the NP-completeness of reachability for HMS machines is proven, suggesting the inherent intractability of safety verification.

Before defining the satisfaction of safety requirements on the states of an HMS machine, the notion of a state literal needs to be defined:

Definition 17 (State Literal): A "state literal" is either a state or the negation of a state. The marking M "satisfies" the literal l (written $M \vdash l$) if (a) $l = s$ and $M(s, 0) = T$, or (b) $l = \neg s$ and $M(s, 0) = F$.

The next two definitions indicate, in two simple cases, when a safety requirement R is satisfied by an execution E (written $E \vdash R$). The temporal logic operator "□" stands for "at the present moment, and at all future moments:"

Definition 18 (Simple Safety Property): Let $E = [M_0, M_1, ...]$ be an execution of H, and let $l_1, ..., l_n$ be literals of H. Then "$\Box (l_1 \vee .. \vee l_n)$" is a "simple safety property of H," and is "satisfied" by execution E if, for every $i \geq 0$, there is an l_k such that $M_i \vdash l_k$.

Definition 19 (Simple Deadline Property): Let $E = [M_0, M_1, ...]$ be an execution of H, let $l, l_1, ..., l_n$ be literals of H, and let $d \geq 0$. Then "$\Box(l \rightarrow ((l_1 \vee .. \vee l_n)$ before d))" is a "simple deadline property of H," and is "satisfied" by execution E if, whenever $M_i \vdash l$, there exists an i', $i \leq i' \leq i+d$, and there exists an l_k such that $M_{i'} \vdash l_k$.

Notice that simple safety properties and simple deadline properties are both "safety properties" in the common meaning of the term [La-77]. They are distinguished here because the

representation within an HMS machine will be somewhat different, as will be seen in the following two theorems that relate satisfaction to unreachability:

Theorem 4 (Simple Safety): Let $H = (S, \Gamma_D, \Gamma_N)$ be an HMS machine, and let SAFE be a simple safety property of H: $\Box(l_1 \vee .. \vee l_n)$. Then there is an HMS machine $H' = (S \cup \{s^*\}, \Gamma_D \cup \{\gamma\}, \Gamma_N)$ such that

(a) for every marking M' of H', $\mathfrak{E}(H', M')\uparrow(S \cup \Gamma_D \cup \Gamma_N) = \mathfrak{E}(H, M'\uparrow(S \cup \Gamma_D \cup \Gamma_N))$

(b) for every marking M' of H' such that $M'(s^*, 0) = F$,

s^* unreachable from M' $\Leftrightarrow$ $E\uparrow(S \cup \Gamma_D \cup \Gamma_N) \vdash$ SAFE for all $E \in \mathfrak{E}(H', M')$.

Proof (sketch): Add to H' the new state s^*, and the following deterministic transition γ:

γ: ( ) $((\neg l_1, 0) \ldots (\neg l_n, 0))$ --> (s^*) [where $\neg l_i = s_i$ whenever $l_i = \neg s_i$].

Property (a) holds, since γ has no primaries or consequents in S. Property (b) holds since s^* can become marked if and only if γ can become enabled, and any execution leading to a marking which first enables γ will fail to satisfy SAFE. □

Theorem 5 (Simple Deadline): Let $H = (S, \Gamma_D, \Gamma_N)$ be an HMS machine, and let DLINE be a simple deadline property of H: $\Box(l \rightarrow ((l_1 \vee .. \vee l_n)$ before d)). Then there is an HMS machine $H' = (S \cup \{s^*\}, \Gamma_D \cup \{\gamma\}, \Gamma_N)$ such that

(a) for every marking M' of H', $\mathfrak{E}(H', M')\uparrow(S \cup \Gamma_D \cup \Gamma_N) = \mathfrak{E}(H, M'\uparrow(S \cup \Gamma_D \cup \Gamma_N))$

(b) for every marking M' of H' such that $M'(s^*, 0) = F$,

s^* unreachable from M' $\Leftrightarrow$ $E\uparrow(S \cup \Gamma_D \cup \Gamma_N) \vdash$ DLINE for all $E \in \mathfrak{E}(H', M')$.

Proof (sketch): Add to H' the new state s^*, and the following deterministic transition γ:

γ: ( ) $((l, -d), ((\neg l_1, [-d, 0]) \ldots (\neg l_n, [-d, 0]))$ --> (s^*)

The demonstration that (a) and (b) hold for H' follows the proof of Theorem 4. □

Since the satisfaction of safety properties is reducible to HMS state unreachability, it is worth noting that the HMS state reachability problem is NP-complete. The proof mimics a proof in [De-88], which demonstrated the NP-completeness of certain planning problems.

Theorem 6 (NP-Complete): The HMS state reachability problem is NP-complete.

Proof (sketch): Reachability is in NP, since a legal firing sequence can be guessed and checked efficiently. NP-hardness follows by reduction of the NP-complete problem 3SAT, the satisfiability of conjuncts of disjuncts of triples of literals [Ga-79]. For any such logical expression, an HMS machine can be constructed which (a) non-deterministically chooses a valuation for all atoms in the first clock tick, and (b) deterministically evaluates the expression at that valuation over the next two clock ticks. The state corresponding to the truth of the expression is reachable if and only if the original expression is satisfiable. □

5. Railroad-Crossing Example

This section illustrates the application of the transformational approach of this paper for verifying safety properties of real-time systems for the example of a railroad-crossing sys-

tem adapted from [Ja-88]. Following our basic approach, the proof is in two parts: (1) In Section 5.1, we represent the deadline-dependent safety property associated with the railroad-crossing system using new states in its HMS specification, following the method of Section 4. (2) In Section 5.2, we demonstrate that the state corresponding to the key safety property is unreachable, using the transformational method of Section 3.

It is important to emphasize that our approach, unlike the method used in [Ja-88] to verify the same safety property is not a mechanical decision procedure. When the system under consideration is complex, there may be no practical mechanical procedure, since the complexity of most methods grows explosively with the size of the system. In contrast, a judicious choice of transformations can, in a few steps, drastically prune a problem which would otherwise be overwhelming. Note that the argument for heuristic proof methods is strengthened by the NP-completeness result of Section 4.

5.1. Requirement Representation for the Railroad-Crossing Example

A railroad-crossing system can be modeled as the interactions of a train and a gate. When the train nears the crossing, a signal is sent to the gate that it should not be in the up position, and when it leaves the crossing a signal is sent indicating that the gate can be up. This system is specified by the HMS machine RR = (S, Γ_D, Γ_N), where

S = {BEFORE-CROSS, NEAR-CROSS, IN-CROSS, PAST-CROSS,
Gate_Up=T, Gate_Up=F, MOVE-UP, UP, MOVE-DOWN, DOWN}

Γ_D = {t_1: (MOVE-UP) ((GateUp=F, 0)) --> (MOVE-DOWN)
t_2: (DOWN) ((GateUp=T, 0)) --> (MOVE-UP)
t_3: (UP) ((GateUp=F, 0)) --> (MOVE-DOWN)}

Γ_N = {t_4: (BEFORE-CROSS) ( ) --> (NEAR-CROSS)
t_5: (NEAR-CROSS) ((NEAR-CROSS, [-300, 0])) --> (IN-CROSS)
t_6: (IN-CROSS) ( ) --> (PAST-CROSS)
t_7: (PAST-CROSS) ((PAST-CROSS, [-100, 0])) --> (BEFORE-CROSS)
t_8: (GateUp=T) ((NEAR-CROSS, 0)) --> (Gate_Up=F)
t_9: (GateUp=F) ((PAST-CROSS, 0)) --> (Gate_Up=T)
t_{10}: (MOVE-UP) ((GateUp=T, 0)) --> (UP)
t_{11}: (MOVE-DOWN) ( ) --> (DOWN)}

The first four states of RR indicate where the train is at any moment: well before the crossing, near the crossing, in the crossing, or past the crossing. The last four states indicate where the gate is at any moment: on its way up, fully up, on its way down, or fully down. The middle two states indicate the signal being sent to the gate: the gate should be up, or the gate should not be up. We may use the following abbreviations for the states: {BC, NC, IC, PC, GUT, GUF, MU, UP, MD, DN}.

This system has the following three deadline properties associated with it:

Deadline 1: Transition from GUT to GUF within 50 moments of NC becoming marked.
Deadline 2: Transition from GUF to GUT within 50 moments of PC becoming marked.
Deadline 3: Transition from MD to DN within 50 moments of MD becoming marked.

As shown in Theorem 5, these deadlines can be represented in RR by adding three new states: Missed-Deadline1, Missed-Deadline2, and Missed-Deadline3 (abbreviated MDL1, MDL2, MDL3) together with the following three deterministic transitions:

t_{12}: ( ) ((NEAR-CROSS, -50), (Gate_Up=T, [-50, 0])) --> (MISSED-DEADLINE1)
t_{13}: ( ) ((PAST-CROSS, -50), (Gate_Up=F, [-50, 0])) --> (MISSED-DEADLINE2)
t_{14}: ( ) ((MOVE-DOWN, [-50, 0])) --> (MISSED-DEADLINE3)

Notice that a deadline is missed in the system if and only if the corresponding state in the extended HMS machine becomes marked.

There is also an important non-deadline safety requirement for the railroad-crossing system: the crossing-arm must be down whenever the train is in the crossing:

$\Box(\neg$IN-CROSS $\vee$ DOWN)

By Theorem 4, this simple safety property can be represented by adding the state UNSAFE-CROSS (abbreviated UC), together with the following deterministic transition:

t_{15}: ( ) ((¬DOWN, 0), (IN-CROSS, 0)) --> (UNSAFE-CROSS)

Lastly, the key desirable system property is that the non-deadline safety requirement above is guaranteed to hold if no deadlines are ever violated:

$\Box(\neg$UC $\vee$ MDL1 $\vee$ MDL2 $\vee$ MDL3)

This simple safety property can now be represented by the new state SYSTEM-FAILURE (abbreviated SF), together with the following deterministic transition:

t_{16}: ( ) ((UC, 0), (¬MDL1, 0) (¬MDL2, 0), (¬MDL3, 0)) --> (SYSTEM-FAILURE)

The extended HMS machine RR is shown graphically in Figure 4. Transitions with no primaries are depicted as arrows from crossbars; requirement states are shaded in gray.

5.2. Transformational Proof of the Railroad-Crossing Example (sketch)

The HMS machine depicted in Figure 4 specifies the railroad-crossing system together with timing constraints and important safety properties. If the state SYSTEM-FAILURE is shown to be unreachable, then the corresponding safety property is guaranteed to be satisfied. In fact, the state can be proven to be unreachable, given the following assumptions about the initial marking M_0: (1) exactly one of {IC, PC, BC, NC} is marked by M_0, (2) exactly one of {GUT, GUF} is marked by M_0, (3) exactly one of {MU, UP, MD, DN} is marked by M_0, and (4) M_0 is consistent (Definition 14). The proof makes use of the fact

that {GUT, GUF} and {IC, PC, BC, NC} are both 1-invariants of RR, which follows from the cyclic structure of transitions into and out of those states.

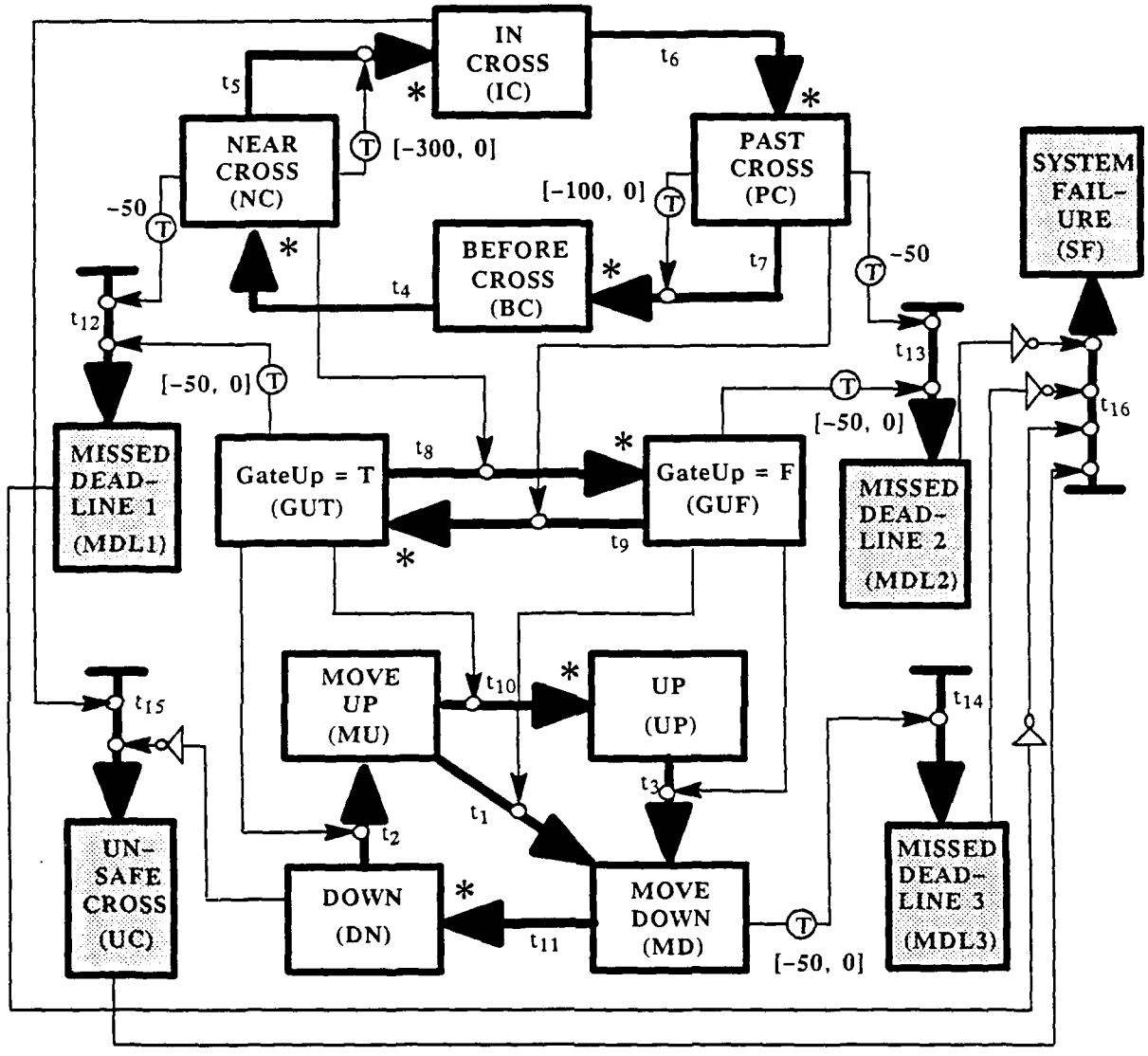

Figure 4. Railroad Crossing with System Requirements

Figure 5 presents an outline of the main steps of the transformational proof. The nodes of this tree show controls on significant transitions generated in the course of the proof, and the values on the branches give the number of transformations from one node to the next. The proof begins by applying a Delay Sharpening transformation to transition t_{16}:

( ) ((¬**MDL1, 0!**) (¬MDL2, 0) (¬MDL3, 0) (UC, 0)) --> (SF)
( ) ((¬MDL1, 0) (¬**MDL2, 0!**) (¬MDL3, 0) (UC, 0)) --> (SF)
( ) ((¬MDL1, 0) (¬MDL2, 0) (¬**MDL3, 0!**) (UC, 0)) --> (SF)
( ) ((¬MDL1, 0) (¬MDL2, 0) (¬MDL3, 0) (**UC, 0!**)) --> (SF)

The first three transitions are then removed (using Transition Deletion #3), since no transition has a deadline state as a primary. Case Split #2 is applied to the fourth transition to add controls (t_{15}, 0) and (¬UC, −1), and Delay Change #3 pushes back the delays on some controls. The transition now has the five controls at the node labeled **A** in Figure 5.

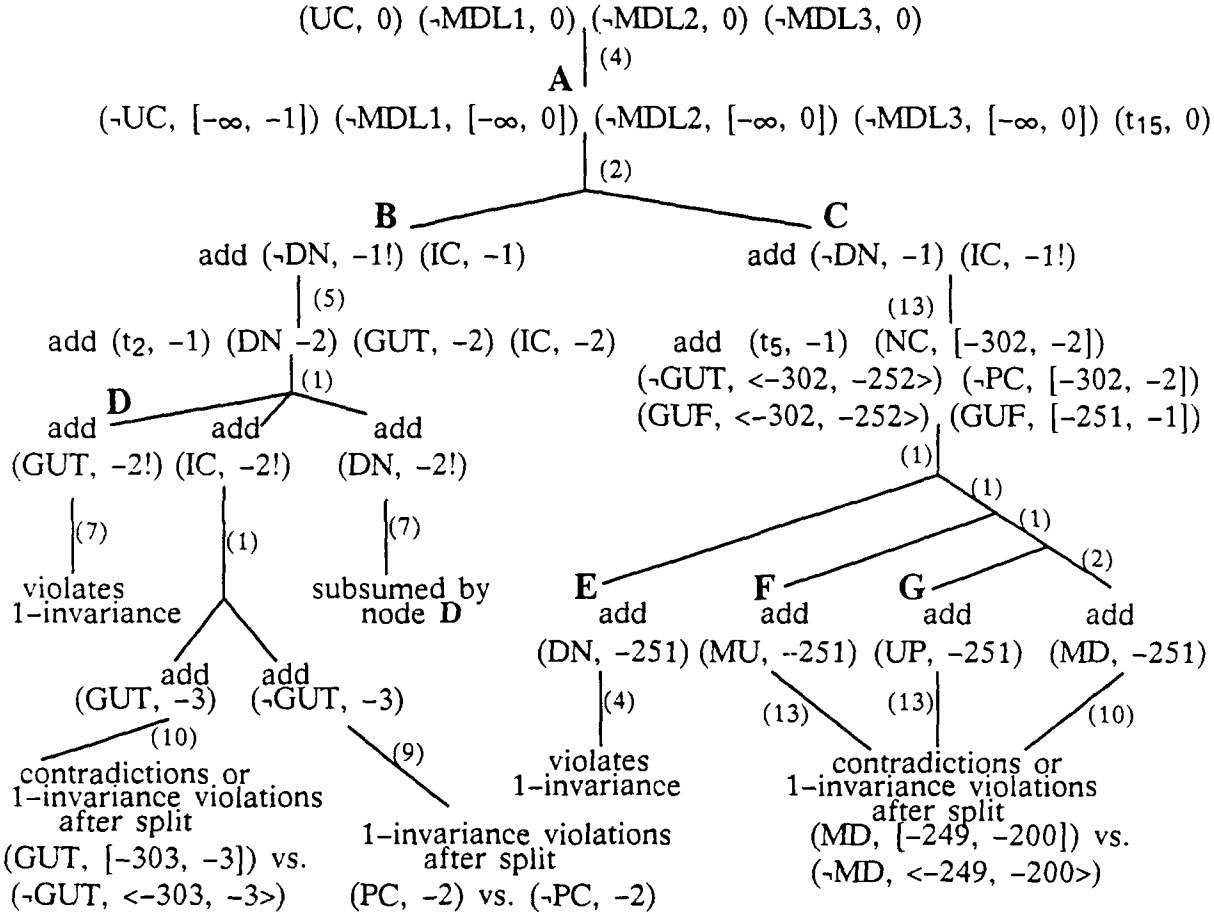

Figure 5. Outline of Transformational Proof for Railroad-Crossing Safety Property

From node **A**, Control Addition #1 supplies controls required for t_{15} to have fired, and Delay Sharpening is applied to those new controls. At this point, the first main branching in the proof occurs (nodes **B** and **C** in Figure 5).

The transition at node **C** is later split into four main cases (using successive Case Split #1's), one of which has the control (DN, −251) added to it (node **E** in Figure 5). From this point, Control Addition #3 adds the control (¬DN, <−250, −1>!), since (¬DN, −1) is already present; then Case Split #3 adds (t_2, <−250, −1>), which is the only way DN can become false; then Control Addition #1 adds (GUT, <−251, −2>), which is necessary for t_2 to fire; and then Transition Deletion #4 eliminates this transition, because (GUT, <−251, −2>) is an invariant conflict with the preexisting control (GUF, [−251, −1]).

There is much repetition in this proof. For example, the paths from node **F** and **G** in Figure 5 take 13 steps, but they agree on their final 11 steps (same transformations involving the

same states and controls). Moreover, there are transformational "cliches" which recur, e.g., the three-step sequence (1) add sometime-change controls X, (2) add transition-based controls Y to satisfy X, (3) add state-based controls Z to satisfy Y. Perhaps such cliches can be replaced by single, high-level transformations.

A total of 104 transformations are applied in the course of this proof, generating a tree with eight main leaves at depths ranging from 19 to 35. Each leaf represents a deleted transition, and, at the end of the proof, no remaining transition has System-Failure as a consequent. By Theorems 1 and 3, then, this state must be similarly unreachable in the original machine, and thus the corresponding safety property holds, i.e., if all specified deadlines are met, the train will never be in the crossing unless the gate is down.

6. Summary and Conclusions

A new formalization was presented for a simple class of HMS machines that are suitable for specifying complex dynamic systems with timing constraints. It was shown how system requirements can be represented within such an HMS machine, thus recasting a question of logical satisfaction as a state reachability problem. A collection of consistent and complete correctness-preserving transformations was given, by which an HMS machine structure can be modified without altering important aspects of its behavior. The combination of requirement representation and structural transformation constitutes a proof method for verifying that a system specification meets its safety requirements, including those involving hard deadlines.

Possible extensions of this work include (1) discovery of more powerful (partial) correctness-preserving transformations, (2) definition of new concepts of partial behavioral equivalence, (3) consideration of other classes of HMS machines, and (4) creation of a user-assisted automated system for applying transformations along the lines of an interactive theorem prover (e.g., [Pa-87]).

References

[Ab-88] Abadi, M. and L. Lamport, "The existence of refinement mappings," *Proc. of the Logic in Computer Science Conference*, Edinburgh, Scotland, July 1988.

[Al-85] Alpern, B. and F. B. Schneider, "Verifying temporal properties without temporal logic," ACM Transactions on Programming Languages and Systems, Vol. 11, No. 1, January 1989, pp. 147-167.

[De-89] Dean, T. and M. Boddy, "Reasoning about partially ordered events," Artificial Intelligence, Vol. 36, No. 3,. October 1988, pp. 375-399.

[Ga-87] Gabrielian, A. and M. E. Stickney, "Hierarchical representation of causal knowledge," *Proc. WESTEX-87, IEEE Expert Systems Conf.*, 1987, pp. 82-89.

[Ga-88] Gabrielian, A. and M. K. Franklin, "State-based specification of complex real-time systems," *IEEE Real-Time Systems Symposium*, 1988, pp. 2-11.

[Ga-79] Garey, M.R. and Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[Hu-85] Huet, G., "Deduction and computation," in *Fundamentals of Artificial Intelligence*, W. Bibel and Ph. Jorrand (Eds.), Springer-Verlag, 1985, pp. 39-74.

[Ja-88] Jahanian, F. and D. A. Stuart, "A method for verifying properties of Modechart specifications," *IEEE Real-Time Systems Symp.*, 1988, pp. 12-21.

[La-77] Lamport, L., "Proving the correctness of multiprocess programs," *IEEE Trans. on Software Engineering*, Vol. SE-3, No. 2, March 1977, pp. 125-143.

[Pa-83] Partsch, H. and R. Steinbruggen, "Program transformation systems," *Computing Surveys*, Vol.15, No.3, September 1983, pp. 199-236.

[Pa-87] Paulson, L., *Logic and Computation*, Cambridge University Press, 1987.

[Ra-59] Rabin, M. O. and D. Scott, "Finite automata and their decision problems," *IBM J. of Research*, Vol. 3, No. 2, pp. 115-125.

[Re-85] Reisig, W., *Petri Nets*, EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer-Verlag, 1985.

[Va-79] Valette, R., "Analysis of Petri nets by stepwise refinements," *J. Comput. Syst. Science*, Vol. 18, 1979, pp. 35-46.

[Vo-89] Vogler, W., "Failure semantics and deadlocking of modular Petri nets," *Acta Informatica*, Vol. 26, 1989, pp. 333-348.